COS 533: Advanced Cryptography — Princeton University
Lecture 1 (September 13, 2017)
Lecturer: Mark Zhandry — Scribe: Mark Zhandry

Notes for Lecture 1

1 Basic Cryptography Review

Basic Notation. Essentially every cryptosystem we will see in this course will depend on a security parameter, which we will denote λ. The idea is that increasing λ will provide better security (which we will formalize in a bit). For now, think of λ as the length of the key, though later on we will sometimes allow the key length to be something different than the security parameter.

For the most part, we will not care too much about the precise model of computation. For concreteness, you can take the model of computation to be Turing machines. For randomized/probabilistic algorithms, we will use Turing machines that have access to a random tape.

Cryptographic algorithms will almost always be required to be efficient. Our notion of efficiency will be polynomial time. We will sometimes restrict to deterministic polynomial-time algorithms, and otherwise allow probabilistic algorithms. We will use PPT as shorthand for probabilistic polynomial time.

We say that a function $\epsilon(\lambda)$ is negligible if it goes to zero faster than any polynomial. More precisely, for any constant c, there exists a constant λ_0 such that $\epsilon(\lambda) < \frac{1}{\lambda^c}$ for all $\lambda > \lambda_0$. We will use negligible functions for any quantity that we want to go to zero extremely fast.

Defining Encryption. A (symmetric key or secret key) encryption scheme consists of two algorithms $(\mathsf{Enc}, \mathsf{Dec})$. Enc is a PPT algorithm that takes as input a key and a plaintext, and outputs a ciphertext. Dec is deterministic polynomial time, takes as input a key and a ciphertext, and outputs a plaintext. For correctness, we require that when used with the same key, Dec inverts Enc. More precisely, for all messages m,

$$\Pr[\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m, k \xleftarrow{\$} \{0,1\}^\lambda] = 1$$

Here, the probability is taken over a random k, and any random coins chosen by Enc. Since the probability is 1, this means that for any key and any coins, Dec will always correctly decrypt a plaintext. It is also possible to consider schemes where the probability 1 is replaced with $1 - \epsilon(\lambda)$ for a negligible ϵ.

For security, we want a definition that captures the following:

- Security holds for arbitrary messaages. We want this so that security holds for English (or any other) language, for numerical data, or for any other use case. As an extreme example, maybe the encryption scheme will only be used to encrypt two messages, "`ATTACK AT DAWN`" and "`ATTACK AT DUSK`". We also want security to hold even in settings where the adversary may have some influence over the messages that are sent. For example, an adversary may attack a particular location, and then wait for the adversary to send a message containing the location's name asking for help. To be most conservative, we will therefore give the adversary complete control over messages that are sent.
- We want to allow multiple messages to be sent, even the same message sent multiple times. Note that we want to hide whether the same message was sent again. For example, if Alice sends a ciphertext c to Bob, and then both attack the next morning, the adversary may very well guess afterward that c encrypted encrypts "`ATTACK AT DAWN`." Imagine a few days later Alice wants to send the same message "`ATTACK AT DAWN`". If the adversary can figure out that the same message was sent (for example, if the encryption scheme always maps "`ATTACK AT DAWN`" to c), then the adversary can now guess that an attack will occur the following dawn, and prepare accordingly.
- The adversary may only care about a single bit of information about the plaintext, or even some arbitrary function of the plaintext. We want to design a scheme that works, no matter what piece of information the adversary is interested in. For example, in the "`ATTACK AT DAWN`" vs "`ATTACK AT DUSK`" setting, it is sufficient for the adversary to learn, say, the last character of the plaintext.

We therefore define security as follows. Let A be an adversary. Let $\mathsf{IND\text{-}CPA\text{-}EXP}_b(A, \lambda)$ be the following experiment on A, parameterized by a bit b:

1. A interacts with a challenger, denoted Ch.
2. At first, Ch chooses a random key $k \xleftarrow{\$} \{0,1\}^\lambda$
3. Next, A sends the challenger two messages m_0, m_1. Ch selects and encrypts m_b: $c \leftarrow \mathsf{Enc}(k, m_b)$. Then Ch sends c back to A.
4. A can repeat step 3 as many times as it wishes. We will charge A one unit of time for every time it repeats step 3.
5. Finally, A outputs a guess b' for b. b' is the output of $\mathsf{IND\text{-}CPA\text{-}EXP}_b(A, \lambda)$

Here, IND refers to indistinguishability, meaning that the adversary is trying to distinguish between two experiments, $b = 0$ and $b = 1$. CPA stands for "chosen plaintext attack". This refers to the fact that the adversary is able to choose the plaintexts that get encrypted.

Definition 1 *An encryption scheme* $(\mathsf{Enc}, \mathsf{Dec})$ *is* IND-CPA secure *(in words,* indistinguishable under a chosen plaintext attack*) if, for all PPT adversaries A, there exists a negligible function ϵ such that*

$$| \Pr[1 \leftarrow \mathsf{IND\text{-}CPA\text{-}EXP}_0(A, \lambda)] - \Pr[1 \leftarrow \mathsf{IND\text{-}CPA\text{-}EXP}_1(A, \lambda)] | < \epsilon(\lambda)$$

We will often simply call such a scheme "CPA secure". Intuitively, this definition means that any guess b' the adversary makes is more or less independent of the actual bit b, since the probabilities for any guess under the two experiments are extremely close.

Question: What happens if Enc is a deterministic scheme? Can it possibly be CPA secure?

Sometimes, CPA security is insufficient. For example, there are use cases where the adversary can send arbitrary ciphertexts to Bob, and learn whether Bob was able to successfully decrypt. There may also be settings where the adversary can learn some information about the plaintext underlying ciphertexts it made up. To conservatively model such attacks, we would like to additionally allow the adversary to query the challenger on aribtrary ciphertexts; in response the challeger decrypts and gives the plaintext back to the adversary.

Question: Why can no scheme satisfy this notion of security as defined so far?

To make sure the adversary has no trivial attacks on the scheme, we have to be careful about how we model the attack experiment. Let $\mathsf{IND\text{-}CCA\text{-}EXP}_b(A, \lambda)$ be the following experiment on A, parameterized by a bit b:

1. A interacts with a challenger, denoted Ch.
2. At first, Ch chooses a random key $k \xleftarrow{\$} \{0,1\}^\lambda$
3. Next, A can make one of three kinds of queries:
   - **Challenge Queries.** These are the same as the CPA queries above. A sends the challenger two messages m_0, m_1. Ch selects and encrypts m_b: $c \leftarrow \mathsf{Enc}(k, m_b)$. Then Ch sends c back to A.
   - **CPA Queries.** These are similar to Challenge queries, except they do not depend on the bit b. A simply sends a single message m. Ch encrypts m, obtaining $c \leftarrow \mathsf{Enc}(k, m)$, and sends c back to A. Notice that a Challenge query can be used to simulate a CPA query by setting $m_0 = m_1 = m$. Thus, at this point, the experiment is still effectively identical to the CPA experiment.

- **CCA Queries.** These are *Chosen Ciphertext Attack* queries. A chooses an arbitrary ciphertext c of its choice, and sends c to Ch. First, Ch checks if c was the response to a challenge query. If it is, we know A can win if it learns the decryption of c. Therefore, we have Ch reject such ciphertext. Ch responds with a special failure symbol $\perp$.

  Otherwise, even if c was the result of a CPA query, Ch decrypts c to obtain $m \leftarrow \mathsf{Dec}(k, c)$, and sends m back to A.

4. A can repeat step 3 as many times as it wishes, making arbitrary queries in arbitrary order. We will charge A one unit of time for every time it makes a query.

5. Finally, A outputs a guess b' for b. b' is the output of $\mathsf{IND\text{-}CCA\text{-}EXP}_b(A, \lambda)$

Definition 2 *An encryption scheme* $(\mathsf{Enc}, \mathsf{Dec})$ *is* IND-CCA secure *(in words,* indistinguishable under a chosen ciphertext attack*) if, for all PPT adversaries A, there exists a negligible function ϵ such that*

$$| \Pr[1 \leftarrow \mathsf{IND\text{-}CCA\text{-}EXP}_0(A, \lambda)] - \Pr[1 \leftarrow \mathsf{IND\text{-}CCA\text{-}EXP}_1(A, \lambda)] | < \epsilon(\lambda)$$

We will often simply call such a scheme "CCA secure".

Public Key Encryption. Symmetric key encryption (meaning both sender and receiver use the same key) was the only kind of encryption for centuries. One significant limitation with symmetric key encryption as defined above is that it requires Alice and Bob to have established a shared secret key at some point in time. This would seem to require either meeting in person, or sending a trusted courier with the key.

One of the major discoveries of the last 50 years was a different kind of encryption called Asymmetric key encryption, or public key encryption. The difference here is that the sender and receiver use different keys. Moreover, the sender's key can actually be *public*. This means that even if the adversary learns the encryption key, it still cannot decrypt messages.

Using such a scheme, not Alice and Bob never need to meet in person. Bob generates a secret decryption key and corresponding public encryption key. He then broadcasts the public key *to everyone*. Now Alice, or anyone else for that matter, can send messages to Bob, and only Bob can decrypt.

In more detail, a public key encryption scheme consists of three algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. Gen is a PPT algorithm that takes as input the security parameter (represented in unary as 1^λ so that it runs in polynomial time in λ) and generates a secret key sk and corresponding public key pk. Enc is the same as before, except it uses pk instead of k. Dec is the same as before, except it uses sk instead of k.

The CPA and CCA games above can be modified for public key schemes. The only differences are:

- At the very beginning of the experiment, Ch runs $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\lambda)$ (instead of running $k \xleftarrow{\$} \{0,1\}^\lambda$).
- Ch gives pk to A at the very beginning, before any queries. This captures the fact that the public key is public, and hence known to A.
- When answering encryption or decryption queries, Ch inputs sk or pk into the encryption or decryption algorithm, respectively.